ANALISIS PERBANDINGAN ANTARA CLUSTER OPENMOSIX DENGAN MPI TERHADAP APLIKASI RENDERING POV-RAY

Anton Dwi Laksono, A.Benny Mutiara MQN,Brahmantyo Heruseto

Universitas Gunadarma
Jl. Margonda Raya 100, Depok, 16424
e-mail : amutiara@staff.gunadarma.ac.id,brahm@staff.gunadarma.ac.id

Abstrak

Openmosix sebagai cluster yang berjalan pada level kernel bisa mengalihkan perkerjaan apa saja dan kapan saja tanpa harus aplikasi tersebut dirancang untuk cluster. Dengan menggunakan konsep load-balancing maka openmosix akan mengalihkan kerja server bila server tersebut telah penuh dengan beban. Oleh karena itu povray sebagai aplikasi rendering gambar yang bila merender suatu gambar 3D membuthkan waktu yang cukup lama sangatlah cocok bila berjalan di cluster openmosix tentulah akan sangat membantu proses penyelesain render suatu gambar dengan kualitas yang bagus dan waktu yang singkat. Dalam percobaan dilakukan perbandingan antara penggunaan cluster jenis OpenMosix dengan MPI untuk aplikasi rendering POV-Ray tersebut.

Kata Kunci : render, cluster, OpenMosix, MPI, POV-Ray

1. Pendahuluan

Komputasi paralel merupakan salah satu teknologi paling menarik yang penting sejak ditemukannya komputer elektronik pada tahun 1940-an. Terobosan dalam pemrosesan paralel selalu berkembang dan mendapatkan tempat disamping teknologi-teknologi lainnya sejak Era Kebangkitan (1950-an), Era Mainframe (1960-an), Era Minis (1970-an), Era PC (1980-an), dan Era Komputer Paralel (1990-an). Dengan berbagai pengaruh atas perkembangan teknologi lainnya, dan bagaimana teknologi ini mengubah persepsi terhadap komputer, dapat dimengerti betapa pentingnya komputasi paralel itu. Inti dari *komputasi paralel* yaitu hardware, software, dan aplikasinya. Paralel Prosesing merupakan suatu pemrosesan informasi yang lebih mendekatkan pada manipulasi rata-rata dari elemen data terhadap satu atau lebih penyelesaian proses dari sebuah masalah. Oleh karena itu komputasi paralel dapat diartikan bahwa komputer dengan banyak prosesor yang mampu untuk melakukan paralel prosesing.

Terdapat paradigma software jika kita berbicara mengenai parallel processing. Gaya pemrograman dan cara pikir tradisional terhadap suatu masalah tidak lagi berlaku ketika solusinya dapat diperoleh dengan menggunakan parallel processing. Komputasi paralel adalah sebuah kesempatan untuk merancang ulang pemrograman dari awal, dengan perspektif yang sama sekali baru. Perspektif baru ini menjadi paradigma tersendiri, dimana paralelisme diaplikasikan dalam banyak gaya pemrograman yang berbeda. Paradigma shared-memory lebih berupa cara pemrograman konvensional, Paradigma distributed-memory berupa pemrograman dengan konsep message passing, Paradigma object-oriented berupa pemrograman dengan konsep server, Paradigma data parallel bergantung pada model SIMD (*Single Instruction Multiple Data*), dan Paradigma *functional dataflow* yang memperkenalkan konsep model MIMD (*Multiple Instruction Multiple Data*) tanpa efek samping.

Suatu program paralel memerlukan koordinasi ketika sebuah tugas bergantung pada tugas lainnya. Ada dua macam bentuk koordinasi pada komputer paralel: asynchronous dan synchronous. Bentuk synchronous merupakan koordinasi pada hardware yang memaksa semua tugas agar dilaksanakan pada waktu yang bersamaan dengan mengesampingakan adanya ketergantungan tugas yang satu dengan lainnya. Sementara bentuk asynchronous mengandalkan mekanisme pengunci untuk mengkoordinasikan prosesor tanpa harus berjalan bersamaan.

Unsupported Personality: PCL

2. Cluster openMosix

OpenMosix merupakan tools untuk kernel yang termasuk jenis keluarga unix, seperti linux. Terdiri dari algoritma-algoritma penggunaan sumber daya bersama yang dapat disesuaikan. Open mosix memperbolehkan multiple Uniprocessors (UP) dan Simetric Multiprocessors (SMP) menjalankan kernel yang sama untuk bekerja dalam pekerjaan yang mirip.

Algoritma-algoritma penggunaan sumber daya bersama dirancang untuk merespon langsung bermacam-macam pemakaian sumber daya pada setiap node. Hal ini dicapai dengan memindahkan proses dari satu node ke node yang lain, secara preemsi dan transparan, untuk load-balancing dan untuk mencegah terjadi tumbukan yang berhubungan dengan pertukaran memori. Tujuannya untuk membuktikan performa cluster-wide dan untuk menciptakan multiuser yang sesuai, lingkungan time-sharing untuk mengerjakan aplikasi sequensial dan paralel. Standar runtime dari open mosix adalah computing cluster, dimana sumber daya cluster-wide tersedia untuk setiap node .

Implementasiawal dari open mosix dirancang untuk berjalan di cluster X86/ workstation Berbasis Pentium, Baik UP dan SMP, terhubung dengan standar LAN. Konfigurasi yang mungkin berkisar dari PC cluster yang kecil yang terhubung dengan Ethernet 10 Mbps, sampai dengan sistem dengan performa tinggi dengan sejumlah teknologi yang mutakhir, SMP server yang berbasis pentiumyang terhubung oleh Gigabit LAN, ATM, ataupun Myrinet.

2.1. Teknologi OpenMosix

Teknologi OpenMosix terdiri dari dua bagian :

1. Mekanisme Preemtive Process Migration (PPM)
2. Algoritma untuk penggunaan bersama sumber daya yang dapat disesuaikan.

Keduanya diimplementasikan pada level kernel menggunakan modul yang bisa dipanggil, sehingga pengantarmukaan kernel tetap tidak dirubah. Meskipun mereka benar-benar transparan terhadap level aplikasi.

PPM dapat memindahkan proses apa saja, kapan saja, ke node yang tersedia. Biasanya, Perpindahan berdasarkan pada informasi yang disediakan oleh salah satu dari algoritma penggunaan bersama sumber daya, akan tetapi pengguna dapat mengesampingkan beberapa sistem pengambilan keputusan secara otomatis dan memindahkan proses mereka secara manual.

Setiap proses mempunyai sebuah Unique Home-Node (UHN) dimana proses itu dibuat. Biasanya ini adalah node dimana pengguna log-in. Model tampilan Sistem tunggal dari openMosix adalah cluster CC (Cache Coherent) , dimana setiap proses terlihat bekerja pada UHN masing-masing, dan semua proses dari sesi pengguna membagi lingkungan kerja dari UHN. Proses yang berpindah ke node yang lain menggunakan sumber daya lokal selama memungkinkan, tetapi berinteraksi dengan lingkungan pengguna melalui UHN.

PPM adalah bagian utama untuk algoritma manajemen sumber daya. Sepanjang syarat untuk sumber daya, seperti CPU, atau memori utama dibawah threshold, proses pengguna dibatasi oleh UHN. Saat syarat-syarat melebihi beberapa level threshold, kemudian beberapa proses boleh dipindahkanke node yanglain untuk mengambil keuntungan dari sumber daya yang tersedia lainnya. Tujuan keseluruhan adalah untuk memaksimalkan performa dengan penggunaan yang efisien dari sumber daya jaringan. Granularitas dari distribusi kerja dalam openmsix adalah proses itu sendiriPenguna dapat menjalankan aplikasi paralel dengan memulai proses penggandaan dalam satu node dan memperbolehkan sistem untuk menugaskan proses-proses ini pada node paling terbaik yang tersedia pada saat itu. Jika pada saat pengeksekusian dari proses, sumber daya yang baru tersedia, Algoritma penggunaan sumber daya bersama dirancang untuk memanfaatkan sumber daya-sumber daya yang baru ini dengan kemungkinan penugasan kembali dari proses diantara node. Kemampuan untuk menerima dan melepaskan proses-proses sangatlah penting untuk kemudahan penggunaan dan untuk menyediakan multiuser yang efisien dan lingkungan pengeksekusian time-sharing.

OpenMosix tidak memiliki kontrol pusat atau hubungan master/slave diantara node-node ; tiap node bisa bejalan sebagai sistem yang berjalan secara otomatis, dan openMosix membuat semua keputusan kontrol secara independen. Rancangan ini memperbolehkan konfigurasi dinamis dimana node-node bisa bergabung atau meninggalkan jaringan dengan gangguan yang rendah. Sebagai tambahan hal ini membolehkan skalabilitas yang besar dan memastikan sistem tersebut berjalan dengan baik pada konfigurasi yang besar sama seperti pada konfigurasi yang lebih kecil.Skalabilitas dicapai dengan menggabungkan ketidakaturan dalam algoritma sistem kontrol, dimana setiap titik mendasarkan keputusannya pada pengetahuan parsial tentang ketetapan dari node-node yang lain, dan bahkan tidak mencoba untuk menentukan ketetapan keseluruhan dari cluster atau node yang lain, contohnya, pada probabilitas algoritma penyebaran informasi, setiap node mengirim, pada interval yang reguel informasi tentang sumber daya yang tersedia untuk terpilih secara acak dari node-node ynag lain.Pada saat yang sama Algoritma ini mempertahankan sebuah jendela kecil dengan informasi terakhir yang diterima. Skema ini mendukung penyekalaan, juga penyebaran informasi dan konfigurasi dinamis.

2.2. Algoritma Penggunaan Sumber Daya bersama cluster openMosix

Algoritma penggunaan sumber daya bersama yang utama dari openMosix yaitu load-balancing dan pengantar memori. Kekuatan algoritma load-balancing terus meners mencoba untuk mengurangi perbedaaan load di antara pasangan node-node, dengan memindahkan proses-proses dari loaded yang tinggi ke loaded yang lebih sedikit. Skema ini menyebarkan semua node-node yang mengeksekusi algotritma-algoritma yang sama, dan pengurangan dari perbedaan load ditampilkan secara independen oleh sepasang node. Jumlah prosesor pada setiap node dan kecepatannya merupakan faktor yang penting dalam algorita load-balancing. Algoritma ini menanggapi perubahan-perubahan load dari node-node atau karakteristik runtime dari proses-proses tersebut. Hal ini berlaku selama tidak ada kekurangan yang berarti dari sumber daya yang lain seperti free memori atau slot proses yang kosong.

Terdapat dua algoritma penggunaan sumber daya bersama yang utama dalam OpenMosix yaitu algoritma pengantar memori (pencegahan penipisan) disesuaikan untuk menempatkan sejumlah proses dalam cluster-wide RAM, untuk menghindari segala kemungkinan tumbukan atau penukaran proses-proses. Algoritma ini terpicu ketika sebuah node mulai kelebihan halaman dan kekurangan free memori. Dalam hal ini algoritma mengesampingkan algoritma load-balancing dan mencoba untuk memindahkan sebuah proses ke sebuah node yang mempunyai free memori yang cukup, walaupun perpindahan ini akan menghasilkan load distribution yang tidak seimbang.

Belakangan ini, openMosix, diberikan algoritma baru untuk memilih node yang mana sebuah program yang diberikan harus dijalankan. Model matematik untuk algoritma ini muncul dari lingkungan sebuah penelitian ekonomi. Menentukan lokasi yang optimal untuk sebuah pekerjaan merupakan masalah yang cukup rumit. Kesulitan utamanya adalah ketersediaan sumber daya pada sebuah cluster dari komputer linux yang sangat beragam. Efeknya, penggunaan memori, CPU, komunikasi, dan tidak dapat dibandingkan. Bahkan mereka tidak diukur dalam unit yang sama.

Sumber daya komunikasi diukur dalam bandwidth, memori dalam kapasitas, dan CPU dalam perputaran. Strategi alami, menyeimbangkan sumber daya-sumber daya melewati semua mesin-mesin, tidak didefinisikan secara benar. Algoritma baru yang ditugaskan oleh openMosix sangatlah menarik karena algoritma tersebut mencoba untuk menerima perbedaan-perbedaan ini didasarkan pada prnsip-prinsip ekonomi dan analisis yang ketat.

Kunci utama dari strategi ini yaitu untuk merubah penggunaan total dari beberapa sumber daya yang beragam, seperti memori dan CPU, menjadi "biaya" sejenis yang tunggal. Perintah-perintah tersebut selanjutnya akan ditugaskan ke mesin di mana perintah-perintah tersebut memiliki biaya yang rendah. Sama seperti ekonomi yang berorientasi pasar.

2.3. Masalah yang Membatasi Kinerja pada Lingkungan Paralel

Algoritma paralel mungkin tampak memiliki speedup yang tinggi secara teoritis, tetapi saat diimplementasikan ke dalam system yang nyata memberikan hasil yang lebih rendah. Berikut adalah beberapa masalah yang dapat membatasi kinerja program paralel.

1. Memory Contention

Eksekusi pemroses ditunda saat menunggu untuk memperoleh akses ke bagian memori yang sedang dipakai oleh proses lainnya. Masalah ini muncul bila sebuah shared data dimiliki oleh sejumlah pemroses paralel. Memory contention merupakan masalah yang muncul pada arsitektur shared memory seperti multiprocessor.

2. *Kode sekuensial yang berlebihan*
   Dalam algoritma paralel akan selalu ada bagian kode sekuensiak yang melakukan operasi tertentu, seperti inisialisasi. Kode sekuensial bisa membatasi speedup maksimum yang dapat dicapai.
3. *Waktu pembuatan proses*
   Dalam praktek, pembuatan proses paralel memerlukan sejumlah waktu. Bila proses yang dibuat memiliki durasi yang pendek, overhead pembuatannya menjadi lebih besar dari waktu yang dihemat oleh komputasinya secara paralel.
4. *Delay komunikasi*
   Delay ini muncul karena interaksi antara dua pemroses pasa multikomputer yabg dikakukan melalui pertukaran pesan. Komunikasi bisa jadi dilewatkan melalui banyak pemroses antara pada jaringan komunikasi. Delay komunikasi yang dihasilkan bisa menurunkan kinerja program.
5. *Delay sinkronisasi*
   Pensinkronan proses paralel, berarti ada proses yang dipaksa menunggu proses lainnya untuk keperluan sinkronisasi. Pada beberapa program paralel, delay yang dihasilkan dapat menyebabkan penurunan kinerja.
6. *Ketidakseimbangan beban*
   Pada beberapa program paralel, proses yang melakukan komputasi dibuat secara dinamis dan harus ditempatkan ke pemroses setelah dibuat. Ada kemungkinan beberapa proses menganggur (idle), sementara yang lainnya memiliki banyak proses yang melakukan komputasi.

2.4. Perpindahan Proses Cluster OpenMosix

OpenMosix mendukung perpindahan proses (PPM) preemsi dan benar-benar transparan. Setelah perpindahan, sebuah proses berlanjut untuk berintraksi dengan lingkungannya tanpa memperhatikan lokasinya. Untuk mengimplementasikan perpindahan proses (PPM), proses perpindahan dibagi dalam dua konteks: user context, dimana bisa dipindahkan, dan system context, yaitu tidak tergantung dengan UHN dan tidak boleh dipindahkan.

User context, biasa disebut dengan remote, berisi kode program, stack, data, peta memori, dan register dari proses. Remote meringkas proses ketika bekerja di level user. System context, biasa disebut deputy, terdiri dari penjabaran dari sumber daya dimana proses itu terhubung, dan sebuah kernel-stack untuk pengeksekusian dari kode system pada keseluruhan proses. Deputy meringkas proses ketika bekerja dalam kernel. Deputy menahan bagian dependent dari system context pada proses; karenanya deputy harus tetap berada dalam UHN dari proses. Sementara proses dapat sering berpindah-pindah diantara node-node yang berbeda, deputy tidak pernah berpindah. Pengantarmukaan antara user-context dan system context sudah didefinisikan dengan jelas. Maka dari itu sangat mungkin untuk memberhentikan setiap interaksi antara context-context ini, dan melanjutkan interaksi ini lewat jaringan. Hal ini diimplementasikan pada layer link, dengan jalur komunikasi spesial untuk interaksi.

Waktu untuk perpindahan memiliki komponen tetap, untuk membangun suatu kerangka proses yang baru pada situs remote baru, dan komponen linear, terbagi rata ke sejumlah memory pages untuk ditransfer.

Untuk pengeksekusian sebuah proses dalam OpenMosix, transparansi lokasi dapat dicapai dengan melanjutkan pemanggilan system site-dependent ke deputy pada UHN. Pemanggilan system (system calls) adalah suatu bentuk synchronous dari interaksi antara konteks dua proses. Semua system calls yang dieksekusi oleh proses tersebut diberhentikan oleh link layer situs remote. Jika system call independent terhadap situs maka system tersebut dieksekusi secara local oleh remote. Jika tidak, system call dilanjutkan ke deputy, yang mengeksekusi system call yang mewakili proses dalam UHN. Deputy mengirimkan kembali hasil ke situs remote, dan melanjutkan untuk mengeksekusi user's code.

Bentuk lain dari interaksi antara konteks dua proses adalah pengiriman melalui sinyal dan event proses wakeup, seperti ketika data dari suatu jaringan dating. Event-event ini membutuhkan bahwa deputy menempatkan secara asynchronous dan berinteraksi dengan remote.

Kebutuhan lokasi ini dipertemukan oleh jalur komunikasi di antara mereka. Dengan skenario yang mirip, kernel yang berada pada UHN menginformasikan kepada deputy dari event tersebut. Deputy memeriksa apakah ada suatu tindakan yang perlu diambil, dan jika memang maka menginformasikan kepada remote. Remote memonitor jalur komunikasi untuk laporan dari event asynchronous, seperti sinyal, tepat sebelum memulai eksekusi user-level. Pendekatan cara ini dapat dikatakan kuat, dan tidak terpengaruh bahkan oleh modifikasi atau perubahan yang besar dari kernel tersebut. Hal ini tergantung pada hampir fitur-fitur no machine-dependent dari kernel, dan hal ini tidak menghalangi jalur ke arsitektur-arsitektur yang berbeda.

3. POV-Ray

POV-Ray kependekan dari "Persistence of Vision Raytracer". POV-Ray merupakan sebuah software untuk membuat grafis komputer dengan kualitas yang sangat tinggi. POV-Ray berlisensi freeware, yang berarti setiap orang bisa menggunakannya, dan memperbanyaknya tanpa biaya sedikitpun.

Yang membuat POV-Ray berbeda dari yang lainnya adalah POV-Ray memiliki kekuatan dan kestabilan untuk memuaskan para pengguna yang sangat kompeten dan berpengalaman, meskipun tidak terlalu sulit bagi para pengguna biasa. Tentu saja faktor terpenting adalah kualitas gambar, dan sesungguhnya POV-Ray memiliki hal tersebut.

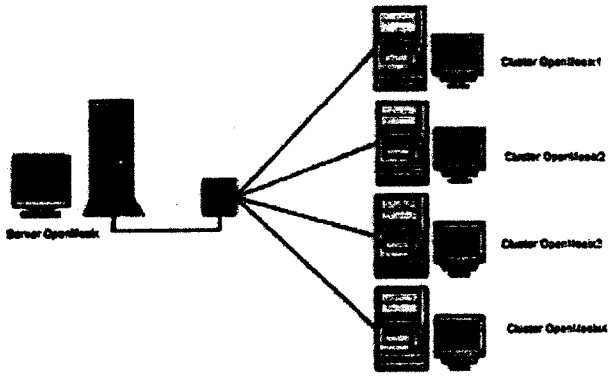
Menurut para pengembang, mereka melihat gambar yang mereka render menggunakan POV-Ray, yang menurut dugaan pertama mereka merupakan fotografi, sangat realistik. Perlu diketahui bahwa foto-realisme merupakan keahlian tingkat tinggi, yang memerlukan banyak latihan untuk menguasainya. Untuk menulis dan memodifikasi scene POV-Ray, pengguna hanya mengedit file teks sesungguhnya yang berisikan perintah - perintah. Pengguna baru mungkin terkejut untuk mempelajarinya, dimana mungkin kedengarannya primitive, hal ini merupakan salah satu fakta yang menunjukkan POV-Ray memiliki kekuatan dan fleksibilitas yang sangat tinggi.

Ada beberapa program untuk merender yang memberikan pengguna fasilitas antar muka point-and-click, tetapi ketika dihadapkan pada keharusan dalam kontrol mutlak suatu scene, hal tersebut sangat sulit jika dibandingkan dengan yang berbasiskan teks, walaupun hal tersebut sangat sulit untuk dipelajari. Hampir setiap orang bisa menggunakan POV-Ray untuk Unix. Jika Anda telah melihat program ray tracing sebelumnya, Anda bisa memiliki kesenangan hanya dengan merender scene contoh dan animasi yang terinsal bersamaan didalam POV-Ray untuk Unix.

Memulai dalam merender POV-Ray beberapa file scene merupakan hal yang sederhana, sesederhana menjalankan "POV-Ray" pada perintah baris dengan diikuti argumen nama filenya. Hal ini akan bekerja baik itu dengan file POV ataupun dengan file INI (selama memiliki hubungan dengan file POV).

4. Pengujian

Dalam pengujian untuk melakukan perbandingan antara cluster OpenMosix dengan MPI dilakukan dengan bentuk jaringan sebagai berikut :



Gambar 1. Arsitektur Jaringan Cluster

Adapun spesifikasi yang digunakan adalah :

Server :

- Processor AMD Duron 1300 MHz On Board Motherboard ECS K7SOM+
- Memori DDR 256 MB
- Hard Disk 40 GB 5400 RPM
- LAN Card 10/100 MBPS Edimex
- Switch 100 MBPS
- VGA On board 32 MB share memory
- Sistem Operasi Linux RedHat 9.0

Client :

- Processor AMD Duron 1300 MHz On Board Motherboard ECS K7SOM+
- Memori DDR 128 MB
- Hard Disk 40 GB 5400 RPM
- LAN Card 10/100 MBPS Edimex
- VGA Onboard 32 MB Share memory
- Sistem Operasi Linux RedHat 9.0

Tabel 1. Pengujian Cluster OpenMosix untuk merender gambar dengan POV-Ray

Banyaknya Pengujian Cluster	Waktu (detik) 2 komputer	Waktu (detik) 3 komputer	Waktu (detik) 4 komputer	Waktu (detik) 5 komputer
1	145	136	125	112
2	144	133	122	112
3	146	132	122	111
4	145	131	123	113
5	145	136	123	114
6	145	135	123	111
7	146	135	124	113
8	146	135	125	114
9	144	135	123	115
10	144	135	123	111
Jumlah	1450	1353	1235	1125
Rata-rata	145	135,3	123,5	112,5

Tabel 2. Pengujian MPI untuk merender gambar dengan POV-Ray

Banyaknya	Waktu	Waktu	Waktu	Waktu

Pengujian Cluster	(detik) 2 komputer	(detik) 3 komputer	(detik) 4 komputer	(detik) 5 komputer
1	135	67	42	31
2	136	69	42	31
3	136	66	42	31
4	136	68	41	32
5	135	66	41	30
6	135	67	42	31
7	134	67	40	31
8	136	68	41	32
9	136	68	41	32
10	135	67	42	31
Jumlah	1354	673	414	312
Rata-rata	135,4	67,3	41,4	31,2

Pada data yang didapat pada tabel 1 di atas, terlihat bahwa dalam merender sebuah gambar dengan aplikasi POV-Ray menggunakan Cluster OpenMosix hasilnya dapat dikatakan cukup signifikan perbedaannya jika dibandingkan dengan menggunakan MPI yang hasilnya dapat dilihat pada tabel 2.

5. Kesimpulan

Dari hasil yang diperoleh dalam pengujian Cluster OpenMosix, maka dapat diambil kesimpulan antara lain:

1. Tidaklah efektif jika menggunakan cluster OpenMosix untuk merender suatu citra apabila dibandingkan dengan MPI, karena cluster OpenMosix menggunakan sistem Distributed Cluster sedangkan untuk MPI menggunakan konsep Paralel Cluster.
2. Server pada cluster Openmosix akan terus bekerja selama server tersebut belum sampai pada titik jenuh dan baru akan berpindah pada client jika sudah jenuh.
3. Pada MPI, beban kerja akan lansung dibagi rata pada server dan client sehingga menjadi lebih efektif.
4. Waktu yang dibutuhkan cluster OpenMosix lebih lama jika dibandingkan dengan waktu yang dibutuhkan MPI untuk merender suatu citra.

6. Daftar Pustaka

[1] Firrar Utdirartatmo, *Pemrograman Paralel dengan PVM di LINUX dan WINDOWS*, ANDI Yogyakarta, 2002.
[2] Anonim, *Pemrograman Paralel di LINUX Berbasis DSM (Distributed Shared Memory)*, ANDI Yogyakarta, 2003.
[3] Instalasi Cluster dengan openMosix ,URL : http://www.clustering.org
[4] Kompilasi dan Konfigurasi Kernel, URL : http://www.kernel.org
[5] Michael J. Quinn, *Parallel Computing : Theory and Practice Second Edition*, 1994.
[6] Mulyadi Santosa, *Clustering di Linux dengan OpenMosix*
Email : A_Mulyadi@telkom.net
[7] POV-ray source code, URL: http://www.povray.org/povlegal.html
[8] Ted G. Lewis & Hesham El-rewini, *Introduction to Parallel Computing*, Prentice-Hall Internasional Editions, 1992.
[9] The userland-tools of the openMosix-system, URL: http://openmosix.sourceforge.net/